Computer Vision and Machine Learning in Industry 4.0: Automatic classification of inserts using image classification with CNN

Prepared by Eduardo Fidalgo Fernández
Modified by Andrés Carofilis, Víctor González and Laura Fernández
Universidad de León (Spain)

Table of contents

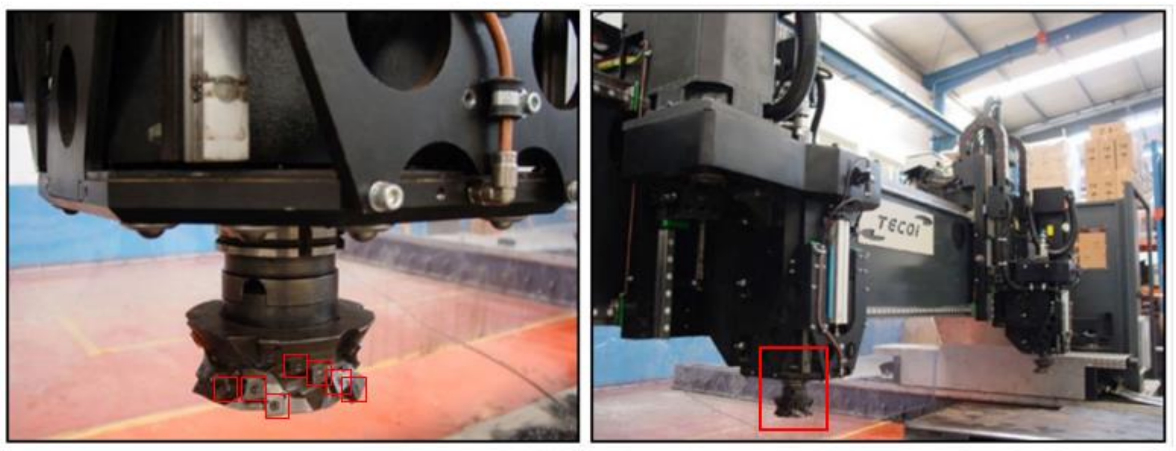
Supported by eit Manufacturing
Funded by the European Union
EIT HEI Initiative

1. Problem to be solved

The equipment we are referencing is related to a project we carried out with the company TECOI.

(https://www.afm.es/es/empresas/asociados/tecoi) (http://www.tecoi.com/)

- **Milling machine with multiple inserts**
- **Inserts wear out after a certain number of operations.**
- **Inserts can break.**
- **Manual inspection after each operation to decide if they are intact, wear out or broken, to replace them before they cause permanent damage to manufactured materials.**



The problem to be solved is the automatic detection of inserts thar are wear out or broken. We will face this problem using Machine Learning, and more specifically, using Image Classification.

2. Objectives

The purpose of this lab is to provide an overview of the main components involved in solving an Image Classification problem using Deep Learning. Specifically, we will use a pretrained Convolutional Neural Network (CNN) trained on the ImageNet dataset (http://www.image-net.org/) to extract features from the images. We will use these features to automatically classify images of inserts in two categories: intact and damaged.

During the process, the student:

- will be guided through the different steps of the pipeline.
- will be asked about the effect of several modifications in the code.
- will check and understand the different parts of a system to classify images based on CNN.

You will work with Google Colab.

<u>NOTE</u>*: This document contains* ***13*** *questions that are optional food for thought.*

Supported by eit Manufacturing
Funded by the European Union
EIT HEI Initiative eit-hei.eu

3. Convolutional Neural Networks

(Some text and the image in the figure taken from http://cs231n.github.io/convolutional-networks/)

In image classification, an image is classified according to its visual content.

In the last years, handcrafted descriptors have been surpassed by the Convolutional Neural Networks (CNN or ConvNet).

CNN are very similar to ordinary Neural Networks: they are made up of neurons that have learnable weights and biases. Each neuron receives some inputs, performs a dot product and optionally follows it with a non-linearity. The whole network still expresses a single differentiable score function: from the raw image pixels on one end to class scores at the other. They also have a loss function (e.g., SVM/Softmax) on the last (fully-connected) layer and all the tips/tricks that were developed for learning regular Neural Networks still apply.

So, what is changing? ConvNet architectures make the explicit assumption that the inputs are images, which allows us to encode specific properties into the architecture. These then make the forward function more efficient to implement and vastly reduce the number of parameters in the network.

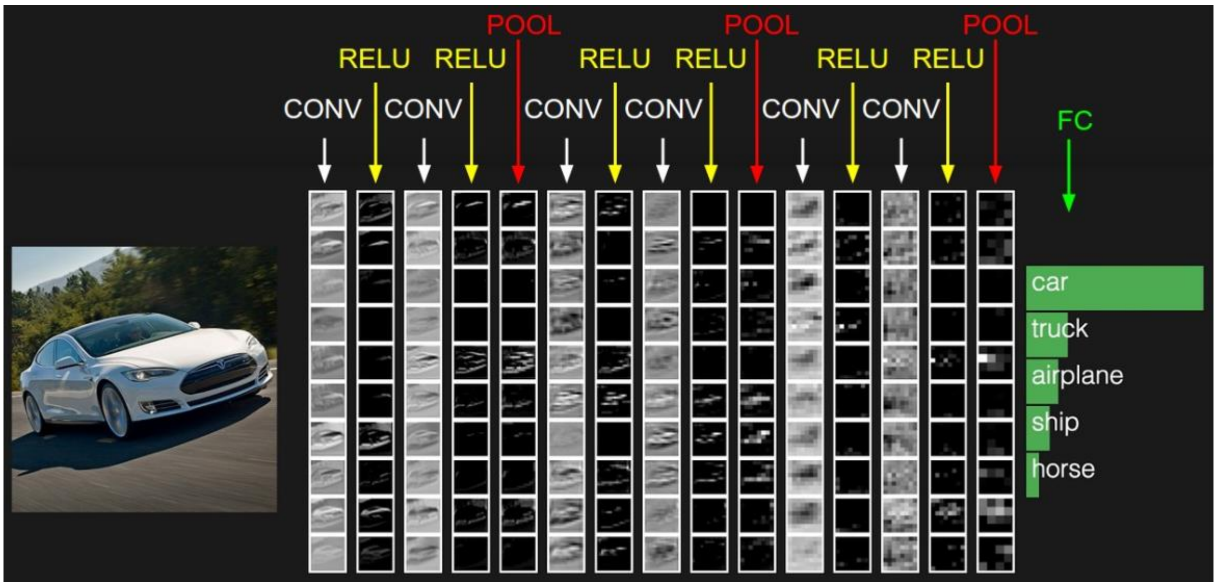


Figure 1: The activations of an example ConvNet architecture. The initial volume stores the raw image pixels (left) and the last volume stores the class scores (right). Each volume of activations along the processing path is shown as a column. Since it's difficult to visualize 3D volumes, we lay out each volume's slices in rows. The last layer volume holds the scores for each class, but here we only visualize the sorted top 5 scores, and print the labels of each one. [Source: http://cs231n.github.io/convolutional-networks/]

In this lab, we will use VGG16 architecture that has the structure shown below:

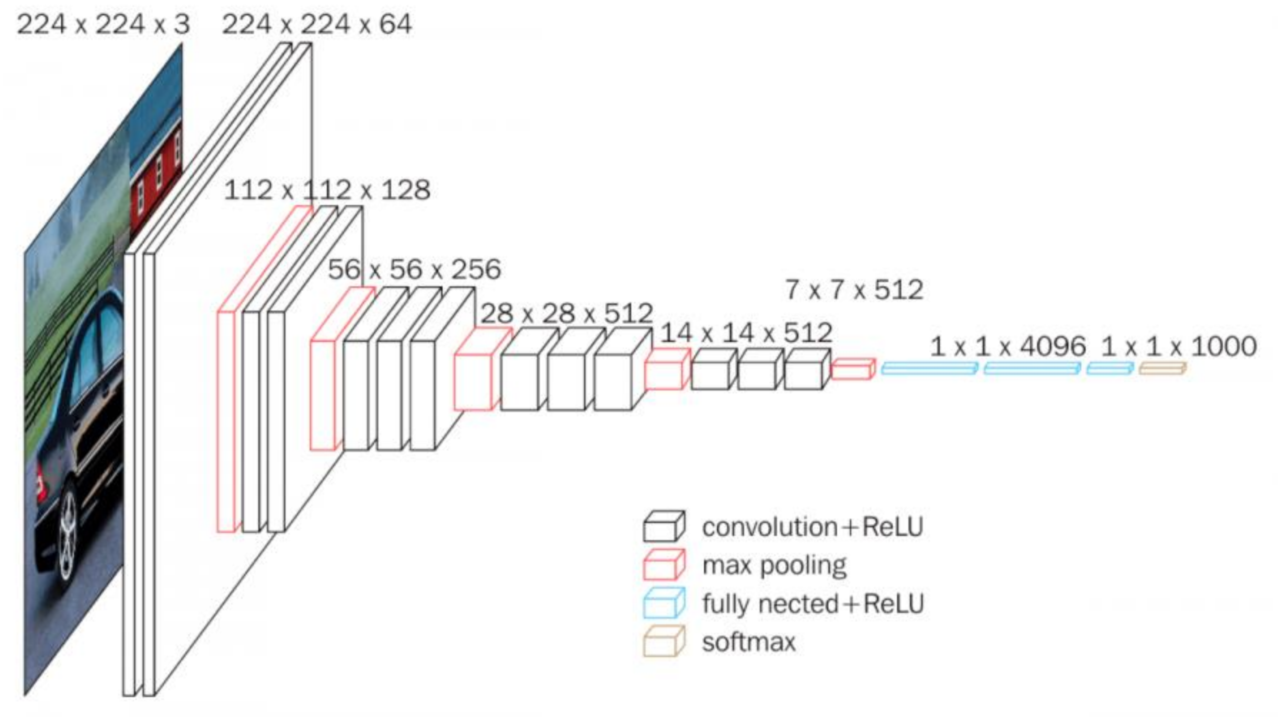


Figure 2: VGG16 architecture [Source: https://neurohive.io/en/popular-networks/vgg16/.]

Typical **activation function**: ReLU: f(x) = max(0, x)

Typical **subsampling by Max-Pooling**

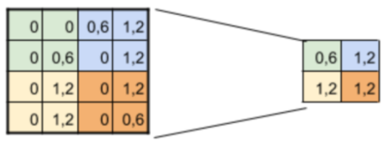


Figure 3: Max-pooling operation

Being the softmax layer a layer that transforms the scores of the different classes into probabilities, as illustrated by the following figure:

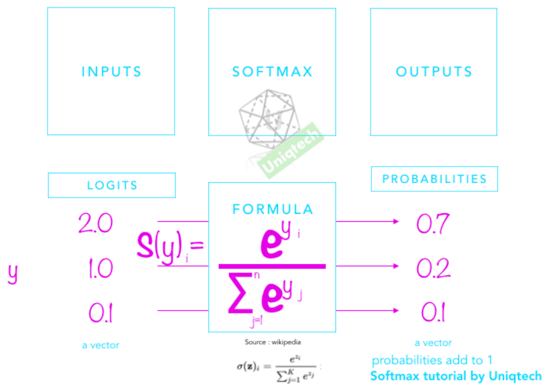


Figure 4: Softmax function [Source: https://medium.com/data-science-bootcamp/understand-the-softmax-function-in-minutes-f3a59641e86d]

4. Keras & Tensorflow

In order to work with CNN, we are going to use Tensorflow (https://www.tensorflow.org), together with Keras (https://keras.io/).

Tensorflow is an open-source library for machine learning, which is developed and used by Google. Keras, is a high-level neural networks API, written in Python and capable of running on top of TensorFlow among others (also CNTK or Theano). It is a deep learning library that allows easy and fast prototyping, provides support for CNN and compatibility with any CPU and GPU.

Tensorflow and Keras will allow us to demonstrate how you can quickly generate a model that is able to detect different categories with reasonably good accuracy.

(From Deep Learning with Python – François Chollet)

Keras (https://keras.io) is a deep-learning framework for Python that provides a convenient way to define and train almost any kind of deep-learning model. Keras was initially developed for researchers, with the aim of enabling fast experimentation.

Keras has the following key features:

- **It allows the same code to run seamlessly on CPU or GPU.**
- **It has a user-friendly API that makes it easy to quickly prototype deep-learning models.**
- **It has built-in support for convolutional networks (for computer vision), recurrent networks (for sequence processing), and any combination of both.**
- **It supports arbitrary network architectures: multi-input or multi-output models, layer sharing, model sharing, and so on. This means Keras is appropriate for building essentially any deep-learning model, from a generative adversarial network to a neural Turing machine.**

In this laboratory session, we will see how Keras could be the key to easily use and apply deep learning to solve a problem related to Industry 4.0: classify inserts into two categories, intact and damaged.

5. Google Colab

Create a folder in your Google Drive and add the file Automatic_classification_of_inserts_using_image_classification_with_CNN.ipynb and the folder »Images_Intact_Damaged_Inserts«.

Into the folder you should have:

- **Automatic_classification_of_inserts_using_image_classification_with_CNN.ipynb**
- **Images_Intact_Damaged_Inserts.zip -> unzip it!**

(and also, the zip file with all the above contents).

Open the file Automatic_classification_of_inserts_using_image_classification_with_CNN.ipynb with Google Colab.

Supported by eit Manufacturing
Funded by the European Union
EIT HEI Initiative

6. Extract CNN features

6.1 Prepare the data

Visually inspect the default dataset provided. Is it easy to visually understand when an insert is damaged (broken or worn) or intact?

6.2 Feature extraction

Based on what we have seen in the lecture related to this laboratory, we are going to do feature extraction with a pre-trained model on ImageNet (http://www.image-net.org/).

As we have indicated, we are going to work with Keras. This framework has some CNN architectures pre-trained on the ImageNet dataset. We are going to work **with the VGG16 architecture**, and we are going to extract features from one of the last layers.

1. **Look for the Keras documentation (https://keras.io/). We are going to look for Models for image classification with weights trained on ImageNet: VGG16. [API docs -> Keras Applications -> Usage examples for image classification models -> Extract features with VGG16]. Add the corresponding line to import and work with VGG16 model (Fill the XXXXX) in the line:**

```
# Import libraries from Keras. Import VGG16
from tensorflow.keras.XXXXXXXXXXXX.XXXXXX import XXXXX, XXXXXX
```

2. **Introduce the name of the folder where the dataset is stored in the variable datasets_available, after the following comment (replace "XX…X" with the name of the folder where the datasets (both image classes) are located).**

```
# To work with multiple datasets. Introduce the name of your dataset(s)
datasets_available = ["XXXXXXXXXXXXXXXXXX"]
```

3. **In the part of the code where the configuration variables are set, replace the *XXX*:**
    a. **set the *model_name* variable to store a string *vgg16*.**
    b. **set the *weights* variable to the ones obtained for ImageNet**
    c. **set a *test set* of 30% of the images (0.30), i.e. 70% will be used for training**

```
for ds in range(0, len(dataset)):
    # Configuration variables config variables
    model_name    = "XXXXX"
    weights       = "XXXXX"
    include_top   = 0
    train_path    = os.path.join(base_dir, dataset[ds])
    features_path = "output/" + dataset[ds] + "/" + model_name + "/features.h5"
    labels_path   = "output/" + dataset[ds] + "/" + model_name + "/labels.h5"
```

Supported by eit Manufacturing
Funded by the European Union
EIT HEI Initiative eit-hei.eu

```
    results      = "output/" + dataset[ds] + "/" + model_name + "/results" + dataset[ds]
+ ".txt"
    model_path    = "output/" + dataset[ds] + "/" + model_name + "/model"
    test_size = XXXX
```

4. **In the part of the code where the pre-trained models are called:**
    a. **Check Keras documentation and set the *function name* to call VGG16 model.**
    b. **Insert the *size of the images* to be fed to the module as 224 x 224.**

```
# Create pre-trained models
    if model_name == "XXXXX":
      base_model = XXXXX(weights=weights)
      model=Model(input=base_model.input, output=base_model.get_layer('fc1').output)
      image_size = (XXX, XXX)
    else:
      base_model = None
```

5. **Run the cell "1. EXTRACT FEATURES". Initially, it will download a file containing the weights of VGG16 that will be used (~550MB). Then, it will start extracting from one of the last layers a feature vector for each image. This feature vector will be used later for training/test a new model. The script will use all the images available and assign them a label [0, 1] automatically.**

6. **While the process takes place, you might answer the following questions:**

***Question_1.* What is the number of layers of this network configuration? https://arxiv.org/pdf/1409.1556.pdf**

Remember that, in all the questions, we are always talking about Configuration D.

***Question_2.* What is the number of parameters of the VGG16 architecture?**

***Question_3.* What is the best top-5 test error (%) obtained by VGG16 in the ILSVRC classification?**

***Question_4.* What is the meaning of top-5 test error?**

Supported by eit Manufacturing
Funded by the European Union
INNOVATION CAPACITY BUILDING FOR HIGHER EDUCATION, PART OF EIT HEI Initiative, eit-hei.eu

6.3 Model training and evaluation

Fill the gaps, i.e. “XXXX” in the cell “2. TESTING PHASE” and execute it. The test set to be used is 30%.

The execution of this script is (almost) immediate.

Then, answer the following questions:

Question_5. What is the algorithm used for training the model? With what kernel?

Question_6. What is the accuracy of the system using VGG16 deep features? Note: instead of printing them through the console, they are saved in a text file.

Question_7. The previous accuracy is the average of how many independent experiments? I.e., How many times the experiments run?

Question_8. What is the size of the feature vector used for training and test?

6.4 Modify the architecture to use MobileNet

Using Keras documentation, modify the code to use MobileNet architecture. Using MobileNet, extract the features from the layer ‘conv1_relu’.

Hint: in the extract_features.py, add an “else if” with an extra condition for a different value – mobilenet - of model_name.

Hint: do not forget to import the corresponding library.

Run the code. You will notice that feature extraction is faster than with the VGG16 one. However, the training and testing phase are slower due to the number of features used by the model.

While you wait for the results, check the paper https://arxiv.org/pdf/1704.04861.pdf and answer the following questions:

Question_9. What is the purpose of this new architecture MobileNet? E.g., suitable for heavy tasks in high demanding computers or mobile applications?

Question_10. What is the accuracy of MobileNet on ImageNet dataset? Compared to VGG16, which one is better? Consider the results obtained in the paper, not in this lab.

Supported by eit Manufacturing
Funded by the European Union
EIT HEI Initiative eit-hei.eu

Question_11. What is the new accuracy of the system in our dataset?

Question_12 Why do you think MobileNet precision is lower than VGG16's?

Question_13. What is the advantage we have achieved with this different architecture against VGG16? In terms of speed and precision.

Supported by eit Manufacturing
Funded by the European Union
INNOVATION CAPACITY BUILDING FOR HIGHER EDUCATION
PART OF EIT HEI Initiative
eit-hei.eu